Using Multiple Namespaces in CC/PP and UAProf

John Gilbert[1], Mark H. Butler
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2003-31
February 24th , 2003*

E-mail: gilberj@tcd.ie, mark-h_butler@hp.com

CC/PP, UAProf, XML, XML namespaces, namespace aliasing, RDF, QName, RDF Schema; schema versioning

CC/PP and UAProf are two related standards, proposed by the W3C and the Open Mobile Alliance respectively, that allow devices such as PCs or smartphones to communicate their capabilities to devices such as web servers. Both these standards use XML namespaces to describe multiple device vocabularies. UAProf specifies a base device vocabulary whereas CC/PP is totally vocabulary agnostic. This paper explores issues surrounding multiple vocabularies: firstly it describes an appropriate data structure for dealing with profiles using multiple vocabularies. Secondly it describes a technique that can be used to process profiles that use incorrect namespaces. Thirdly it describes issues surrounding the automatic retrieval of schemas from namespace URLs, and issues surrounding the encoding of version information in those URLs. Finally it outlines some techniques that can be used to simplify the problem of dealing with multiple vocabularies and multiple vocabulary versions.

* Internal Accession Date Only
Approved for External Publication
[1] Trinity College Dublin, Eire

Using multiple namespaces in CC/PP and UAProf

John Gilbert (gilberj@tcd.ie)[1]
Mark H. Butler (mark-h_butler@hp.com)
Hewlett Packard Laboratories, Bristol UK
16 January 2003

Abstract

CC/PP and UAProf are two related standards, proposed by the W3C and the Open Mobile Alliance respectively, that allow devices such as PCs or smartphones to communicate their capabilities to devices such as web servers. Both these standards use XML namespaces to describe multiple device vocabularies. UAProf specifies a base device vocabulary whereas CC/PP is totally vocabulary agnostic. This paper explores issues surrounding multiple vocabularies: firstly it describes an appropriate data structure for dealing with profiles using multiple vocabularies. Secondly it describes a technique that can be used to process profiles that use incorrect namespaces. Thirdly it describes issues surrounding the automatic retrieval of schemas from namespace URLs, and issues surrounding the encoding of version information in those URLs. Finally it outlines some techniques that can be used to simplify the problem of dealing with multiple vocabularies and multiple vocabulary versions.

Keywords

CC/PP, UAProf, XML, XML namespaces, namespace aliasing, RDF, QName, RDF Schema, schema versioning.

1 Introduction

CC/PP (Composite Capabilities / Preference Profiles) is a proposed W3C standard[1] for the transmission of information about device capabilities. A CC/PP profile is a Resource Description Framework (RDF) model[2], written in its XML serialisation as a two level structure consisting of a number of structural elements called components, each of which contains descriptive elements called properties. UAProf is a specific variant of CC/PP that also supplies a standard base vocabulary. When a profile is sent as part of a request to a CC/PP enabled server, the server may use the information contained within the profile to adapt web content to the target device. A sample UAProf profile is shown in Figure 1 and its RDF/XML serialisation is shown in Figure 2.

An XML namespace[3] is a collection of XML element and attribute names, used in an XML document, associated with a particular URI[4]. Namespaces are often used to provide vocabularies, where the elements and attributes have an associated conceptualization used in many XML document instances. In CC/PP a vocabulary is

[1] John Gilbert was a summer intern at HP Labs Bristol from Trinity College Dublin, Eire.

typically aimed at a specific type of device or a specific use case and is normally specified using RDF Schema[5]. For example, the UAProf vocabulary maintained by the OMA[6] (formerly the WAP Forum) is targeted at describing the capabilities and preferences of an Internet enabled mobile phone and its user.

An RDF model is a series of statements about resources where each statement is represented by a triple consisting of a subject, a properties and an object. Resources and properties in RDF are identified by a QName, short for qualified name, which contains a namespace identifier along with a name from the namespace.

In a CC/PP profile it is common to see two or more XML namespace declarations at

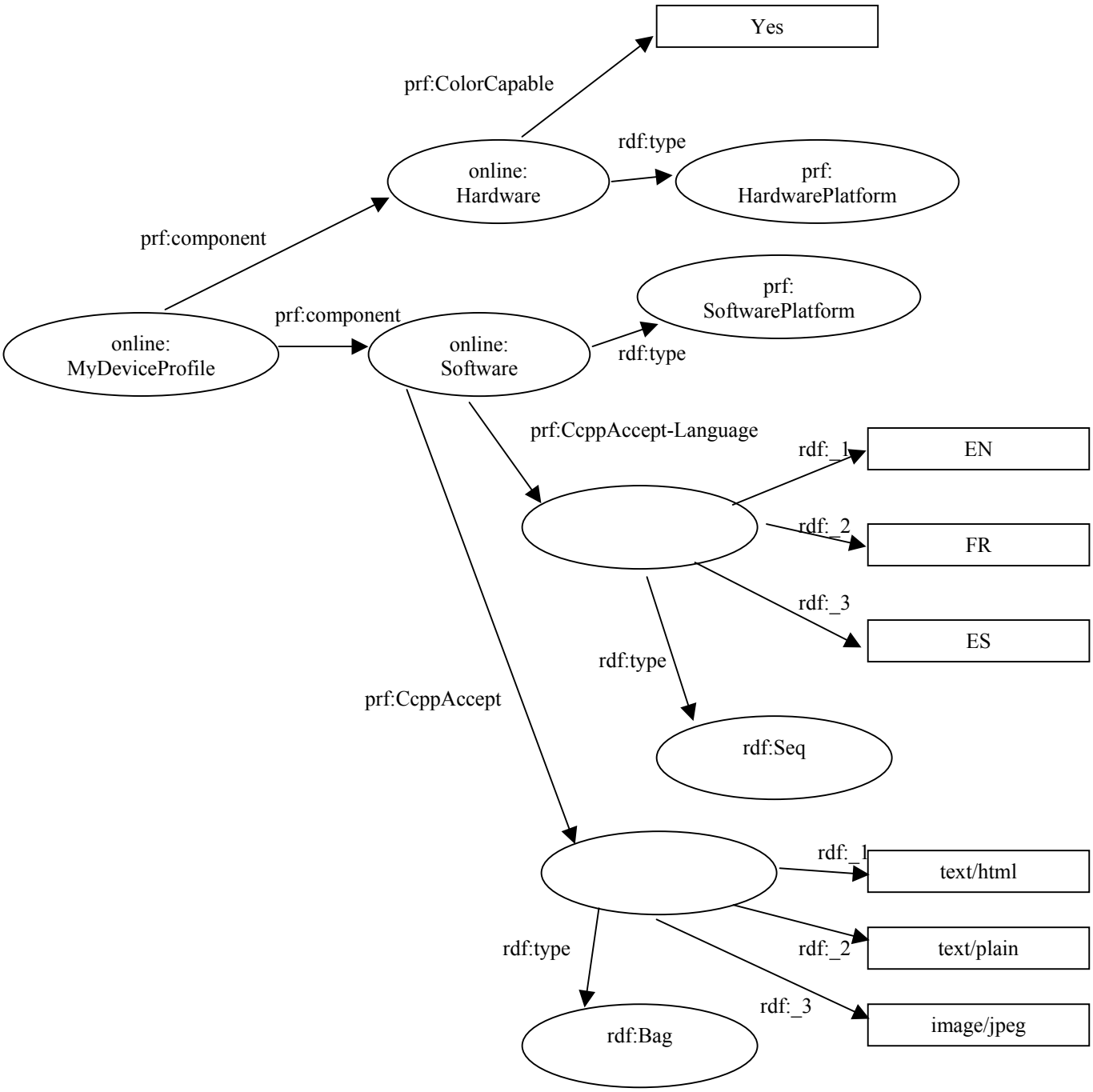

Figure 1 -Example UAProf profile represented in RDF

```
<?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-20010430#">
  <rdf:Description rdf:ID="MyDeviceProfile">
  <prf:component>
   <rdf:Description rdf:ID="HardwarePlatform">
    <rdf:type
 rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
 20010430#HardwarePlatform"/>
    <prf:ColorCapable>Yes</prf:ColorCapable>
   </rdf:Description>
  </prf:component>
  <prf:component>
   <rdf:Description rdf:ID="SoftwarePlatform">
    <rdf:type
 rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
 20010430#SoftwarePlatform"/>
    <prf:CcppAccept>
     <rdf:Bag>
      <rdf:li>text/html</rdf:li>
      <rdf:li>text/plain</rdf:li>
      <rdf:li>image/jpeg</rdf:li>
     </rdf:Bag>
    </prf:CcppAccept>
    <prf:CcppAccept-Language>
     <rdf:Bag>
      <rdf:li>EN</rdf:li>
      <rdf:li>FR</rdf:li>
      <rdf:li>ES</rdf:li>
     </rdf:Bag>
    </prf:CcppAccept-Language>
   </rdf:Description>
  </prf:component>
 </rdf:Description>
</rdf:RDF>
```

Figure 2-Example UAProf profile serialized in RDF/XML

the top of the profile. These indicate that the profile uses RDF, that the profile uses CC/PP and that the profile uses one or more specific vocabularies. Many UAProf profiles, although based on CC/PP, omit the CC/PP namespace and just include a UAProf vocabulary namespace indicating that the profile uses the UAProf vocabulary and structure.

The Universal Resource Identifier (URI) used to identify each namespace may be a valid Universal Resource Locator (URL). If it is a valid URL, it may refer to a retrievable document that contains more information about the vocabulary. In the case of CC/PP, if such a document is available it is most likely to be represented using RDF Schema. However there is no guarantee or requirement that such a document exists.

As already noted, a CC/PP profile does not solely consist of properties: it also has an additional level of structure called components. QNames are used to represent both components and property names. As CC/PP uses QNames, this allows us to distinguish between two or more properties (or components) with the same name from different vocabularies within a single profile. This is necessary because a CC/PP

Namespace: http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#

Attribute Name (Primary Key)	*Attribute Value(s)*	*Parent Component*	*Value Data Type*	*Value Collection Type*	*Resolution Rule*
ColorCapable	Yes	HardwarePlatform	Boolean	Simple	Override
CcppAccept	text/html text/plain image/jpeg	SoftwarePlatform	Literal	Bag	Append
CcppAccept-Language	EN FR ES	SoftwarePlatform	Literal	Sequence	Append

Figure 3-Example UAProf profile data structure

profile may be created using multiple vocabularies, which may contain overlapping local names.

1.1 Overview of this report

While implementing CC/PP and UAProf in DELI[7], a CC/PP API for servers, a number of issues have been identified regarding the use of namespaces within CC/PP profiles and their associated vocabularies. This report aims to describe these issues and provide an explanation of how they have been addressed in the most recent version of DELI.

Specifically firstly it describes an appropriate data structure for dealing with profiles using multiple vocabularies. Secondly it describes a technique that can be used to process profiles that use incorrect namespaces. Thirdly it describes issues surrounding the automatic retrieval of schemas from namespace URLs, and issues surrounding the encoding of version information in those URLs. Finally it outlines some techniques that can be used to simplify the problem of dealing with multiple vocabularies and multiple vocabulary versions.

2 Issues

2.1 Representing properties and components using QNames

Early versions of DELI were designed to process UAProf profiles. As UAProf enabled devices typically send profiles that use a single UAProf vocabulary, this meant that a UAProf profile can be represented as a list of CC/PP properties, each associated with a property value or set of values and a set of vocabulary attributes such as parent component or resolution rule as shown in Figure 3. Component and property names within profiles could be represented using local names i.e. HardwarePlatform and ColorCapable. Namespace information on the other hand was associated directly with the profile.

However although today profiles generally only use a single vocabulary and hence a single namespace, it became apparent that such an approach could not cope with two

Attribute Name (Primary Key)	**Parent Component (Primary Key)**	*Attribute Value(s)*	*Value Data Type*	*Value Collection Type*	*Resolution Rule*
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#ColorCapable	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#HardwarePlatform	Yes	Boolean	Simple	Override
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#CcppAccept	http://www.wapforum.org/profiles/UAPROF/ccppschema-20010430#SoftwarePlatform	text/html text/plain image/jpeg	Literal	Bag	Append
http://www.wapforum.org/profiles/UAPROF/ccppschema-	http://www.wapforum.org/profiles/UAPROF/ccppschema-	EN FR ES	Literal	Sequence	Append

Figure 4 - Revised UAProf profile data structure

situations that are possible in CC/PP: firstly, in CC/PP, it is possible to use multiple vocabularies in the same profile. Secondly it is also legal to have multiple versions of the same property in different components.

In order to support multiple vocabularies in a single profile, DELI was modified to represent components and properties as QNames. Additional API methods were created to allow users to retrieve properties as fully qualified QNames as well as using unqualified local names. The former approach would be used when they are referring to a specific vocabulary, whereas the latter approach would be used if the user does not wish to distinguish between different vocabularies. In addition, the API needs to support methods commonly used on QNames such as retrieving the qualifier (the namespace) in order to determine the vocabulary and version of a property or retrieving the fragment i.e. the local name of the property or component.

Due to the use of the component / property structure in CC/PP, it was decided that simply using a single QName was not sufficient to unambiguously identify a property in the profile. As already noted in CC/PP it is valid for a property to appear inside multiple components. For example, we can imagine the situation where a Vendor property might be applied to both a SoftwarePlatform and HardwarePlatform component within a single profile. Thus to uniquely identify a property, it is necessary to use two QNames: one representing the component, and the other representing the property. This representation is shown in Figure 4, where the primary key is now composed of the attribute QName and the component QName. For lookup efficiency, hash tables can be used to map to the primary key, the property QName, the parent component QName and the property and component local names as deemed necessary by the implementer.

2.2 Coping with erroneous namespace usage in profiles

Namespace aliasing is an efficient way of dealing with profiles that use arbitrary namespaces i.e. namespaces that are not defined in the UAProf specifications and hence do not correspond to retrievable schemas. As DELI requires vocabulary information to process a profile it is necessary to guard against this problem in two ways. Firstly DELI can be configured to alias these namespaces to an existing one. Secondly it has a well-defined fallback behaviour if it encounters a profile that uses an unknown namespace i.e. assume any unrecognized properties are literals and use the override resolution rule.

In DELI, we load each vocabulary definition file or schema into memory when we initialize the processor. In early versions of DELI, we had to load multiple versions of certain vocabulary definitions in order to cope with the arbitrary namespace problem. Clearly this is inefficient; a better solution is to map namespaces explicitly to vocabulary definitions using a lookup table. In DELI the Vocabulary class stores this table, containing direct mappings from incorrect namespace URIs to the correct namespace URI used internally by DELI. Each time a new profile is processed, the appropriate lookups are performed using this table to ensure the correct URI is used. This table is configured by the namespace alias / vocabulary configuration file. Here is an example of a namespace alias configuration file:

```
<?xml version="1.0"?>
<namespaceConfig>
 <namespaceDeclaration>
  <namespace>
   <uri>http://www.wapforum.org/UAPROF/ccppschema-20000405#</uri>
   <schemaVocabularyFile>
    config/vocab/ccppschema-20000405.rdfs
   </schemaVocabularyFile>
  </namespace>
  <namespace>
   <uri>http://www.wapforum.org/UAPROF/ccppschema-20010330#</uri>
   <aliasUri>
    http://www.wapforum.org/profiles/UAPROF/ccppschema-20010330#
   </aliasUri>
   <aliasUri>
    http://www.wapforum.org/UAPROF/ccppschema-19991014#
   </aliasUri>
   <schemaVocabularyFile>
    config/vocab/ccppschema-20010330.rdfs
   </schemaVocabularyFile>
  </namespace>
 </namespaceDeclaration>
</namespaceConfig>
```

will be represented within DELI's vocabulary namespace lookup table as:

URI used by profiles	URI used by DELI internally
http://www.wapforum.org/UAPROF/ccppschema-20000405#	http://www.wapforum.org/UAPROF/ccppschema-20000405#
http://www.wapforum.org/UAPROF/ccppschema-20010330#	http://www.wapforum.org/UAPROF/ccppschema-20010330#
http://www.wapforum.org/profiles/UAPROF/ccppschema-20010330#	http://www.wapforum.org/UAPROF/ccppschema-20010330#
http://www.wapforum.org/UAPROF/ccppschema-19991014#	http://www.wapforum.org/UAPROF/ccppschema-20010330#

2.3 Automatic retrieval of schemas

As the previous section noted, in order to process CC/PP and UAProf profiles we need additional information about the vocabulary in use. Currently processors are preconfigured for vocabularies during initialization, but it is desirable for processors to be able to automatically configure themselves for new vocabularies. However as we have already noted, namespaces URIs are used to uniquely identify something so may not be directly usable for the retrieval of a schema[8]. In addition, even if schemas are available, experience has shown they often contain errors and do not encode sufficient information to describe vocabularies[9]. If a CC/PP processor tries to retrieve schema and fails, it may not be possible for it to gracefully recover so that the server can return content to the target device.

There is a simple solution to this problem: adopt a convention to distinguish between namespaces that indicate retrievable resources. For example we could use a URL i.e. a name starting with `http://` if and only if the URL returns a schema document and a URN (Universal Resource Name) e.g. start with `urn://` to indicate there is no retrievable schema document. For more details of URIs, URLs and URNs see [10].

2.4 URIs and resource versioning

Another problem when dealing with multiple versions of the same vocabulary is there is often version information in a namespace, but that version information is not encoded in a standardised way. Implementers may be tempted to try to use this version information but this will not work reliably across different vocabularies. For example versioning in UAProf is done like this:

```
http://www.wapforum.org/UAPROF/ccppschema-20000405#
http://www.wapforum.org/UAPROF/ccppschema-20020530#
```

whereas versioning in CC/PP is done like this:

```
http://www.w3.org/2000/07/04-ccpp#
http://www.w3.org/2002/09/24-ccpp#
```

In addition there are other alternative approaches to versioning URIs such as the Tag URI scheme[11]. Clearly the problem of how to identify evolving resources is difficult, but this matter is currently under consideration by the W3C Technical Architecture Group (TAG)[12] to see if there is any need for a generalised solution. We would like to propose that resources on the World Wide Web should be identifiable via identity and version. Having a clear separation of these axes allows user agents to easily determine the set of available versions of a resource, and choose specific versions of a resource as well as the most recent version. The ability to do both is essential for the automatic configuration of a constantly evolving web, while maintaining the principle of invariance of web resources identified in [13].

2.5 Dealing with multiple vocabularies and vocabulary versioning

As already noted, CC/PP may deal with different vocabularies aimed at different device types or different use cases. For each of these vocabularies, there may be several different versions of the same vocabulary. Currently it is not clear what is the best way to process multiple vocabularies, but it is possible to identify some strengths and weaknesses of some possible approaches.

For example the approach to vocabulary versioning currently used in UAProf involves rewriting the vocabulary schema for each version and publishing it under a new namespace URI. Therefore a new version is released every time an update to the vocabulary is made, and currently there are three versions available from the OMA website. This approach has the advantage that typically a profile only uses one namespace i.e. a specific version of the vocabulary, simplifying the creation of profiles. It also means that UAProf processors can adopt a simpler processing model as described previously.

However although this approach greatly simplifies processing profiles, it has some disadvantages when you begin to consider multiple versions of vocabularies, as vocabulary properties are duplicated every time a new version of the UAProf vocabulary is released. As these properties are associated with a new QName, it may be up to the application developer to determine that these properties have the same meaning. In addition it is possible servers may encounter requests that mix vocabulary versions, for example when the reference profile used by a device uses one version whereas an intermediate proxy server uses another version. Ideally the CC/PP processor should hide these complexities from the developer where possible.

There are five possible solutions to this problem: ignoring namespace information, extending the capability class method for profile matching, denoting equivalence using an external file, alternative approaches to versioning vocabularies and core device attributes. The following sections describe each of these approaches in more depth.

2.5.1 Ignoring Namespace Information

The first technique is to ignore namespace information altogether i.e. just work with local names rather than QNames. This automatically hides the complexities of processing multiple versions of UAProf from the developer. This approach has the advantage it will require little, if any, extension of existing processors. Therefore DELI does support this approach as it provides methods to retrieve properties via local names as well as QNames.

However this does not solve the problem of how to merge profiles that use different versions of the same vocabulary in a single request. To understand this problem, it is necessary to consider why new vocabularies are introduced. There are two reasons: firstly to add new properties to a vocabulary. Secondly there may be changes to existing properties i.e. changes in data type, collection type and parent component of the property. Therefore when merging profiles that use different vocabulary versions we have to consider the vocabulary on a property-by-property basis. If the property is unchanged, we can treat both versions in the same way. If the property has changed, we either treat the two versions as distinct properties or we treat them the same but adopt a policy for deciding which version to adopt.

In UAProf typically the later version of the property supersedes the earlier version as changes to data type, collection type, parent component or resolution rule have been introduced to correct errors in the original vocabulary. For example the property SecuritySupport has been changed from a Simple Literal to a Bag Literal in later versions of the vocabulary. This is because phones may support more than one

Security protocol. Therefore if a request contains multiple versions of SecuritySupport associated with different vocabulary versions, we might want to treat them all as Bag Literals. This process is not yet implemented in the current version of DELI.

Therefore although the local name approach can deal with versioned vocabularies, it fails if the processor is likely to encounter two distinct vocabularies that use the same property name but with different meanings. In addition it cannot cope if a property changes name between different versions of a vocabulary, for example from `WapPushMsgSize` to `PushMsgSize` as it is using the name (i.e. fragment ID) to infer equivalence.

2.5.2 Supporting multiple vocabularies in capability classes

A previous report[14] outlined a mechanism called capability classes that simplify the task of matching profiles to resources by providing a method of describing constraints that must be met in order to use a particular resource. For example the following capability class file

```
<?xml version="1.0" encoding="UTF-8"?>
<classes>
  <class name="smallScreen">
    <or>
     <lessthan value="160x160">ScreenSize</lessthan>
     <lessthan value="20x20">ScreenSizeChar</lessthan>
    </or>
  </class>
  <class name="largeScreen">
    <or>
     <greaterthan value="320x240">ScreenSize</greaterthan>
     <greaterthan value="80x40">ScreenSizeChar</greaterthan>
    </or>
  </class>
  <class name="jpegcapable">
    <contains value="image/jpeg">CcppAccept</contains>
  </class>
  <class name="color">
    <true>ColorCapable</true>
  </class>
  <class name="blackandwhite">
    <not>
     <true>ColorCapable</true>
    </not>
  </class>
  <class name=”colorphone”>
    <and>
     <lessthan value=”90x120”>ScreenSize</lessthan>
     <contains value=”wml”>CcppAccept</contains>
     <true>IsColorCapable</true>
    </and>
  </class>
</classes>
```

defines four capability classes: *smallScreen*, *largeScreen*, *jpegcapable* and *color*. In the case of *smallScreen*, the device only belongs to this class if it has a screen smaller than 160 wide and 160 pixels high or if it has a screen that is smaller than 20 characters wide and smaller than 20 characters high. Alternatively a device meets the *jpegcapable* capability class criteria if it can display the MIME type image/jpeg. This section outlines how to extend capability classes to provide support for multiple namespaces.

In order to do this capability classes need to be able to both treat properties from different namespaces identically and distinguish between them. For example in the UAProf vocabulary, many properties are unchanged between different versions so may be matched and treated the same when evaluating constraints. Alternatively properties may have changed name or parent component, so we may want to use the OR expression along with several capability operands in order to treat different properties in the same way.

In the revised version of capability classes proposed here, we still have the option of ignoring namespace information when defining a capability class. For example the following capability class definition

```
<class name="jpegcapable">
    <contains value="image/jpeg">CcppAccept</contains>
</class>
```

says that any profile that has a property with local name CcppAccept regardless of the namespace used with the value image/jpeg matches the jpegcapable capability class. Alternatively we may want to define a capability class that explicitly states that the constraint must match a specific namespace:

```
<class name="jpegcapable">
  <contains value="image/jpeg"
            namespace="http://www.wapforum.org/UAPROF/ccppschema-20000405#">
   CcppAccept
  </contains>
</class>
```

Here the profile only belongs to the jpegcapable class if it has a CcppAccept property with value image/jpeg in a specific UAProf namespace.

In addition to treating namespaces in an identical fashion, or specifying a specific namespace, we may also wish to treat a subset of namespaces in an identical fashion. To simplify this situation we also provide a mechanism similar to the namespace aliasing mechanism proposed previously. Here we alias a number of namespaces to an abbreviated namespace name that can be used in a capability class definition, in a similar way to the use of abbreviations for namespaces in XML. For example we might want to define a number of namespaces as having the abbreviated namespace UAPROF:

```
<vocabularyAlias>
 <aliasName>UAPROF</aliasName>
 <aliasList>
  <li>http://www.wapforum.org/UAPROF/ccppschema-20000405#</li>
  <li>http://www.wapforum.org/UAPROF/ccppschema-20010330#</li>
  <li>http://www.wapforum.org/UAPROF/ccppschema-20020530#</li>
 </aliasList>
</vocabularyAlias>
```

This abbreviation can then be used in a capability class definition file e.g.

```
<class name="jpegcapable">
  <contains value="image/jpeg"
            namespaceAlias="UAPROF">CcppAccept</contains>
</class>
```

In the example above, a device only belongs to the jpegcapable class if it has the property CcppAccept in one of the UAProf namespaces defined previously with the value 'image/jpeg'.

The mechanisms outlined above allow capability classes to process multiple vocabularies. However this approach has two limitations: firstly it is only applicable to CC/PP, and cannot be applied to RDF vocabularies in general. Secondly this approach still relies on the author of the capability class file to understand the different vocabularies in use, and the variations between them. This is better than every application developer or content author having to be familiar with the complexities of all the device vocabularies in use, but it is still not ideal. A better solution would be to provide mechanisms that allow organisations creating vocabularies to hide some of the variations between different vocabulary versions from users. The next section explores the use of Semantic Web techniques to achieve this goal.

2.5.3 Denoting equivalence between vocabularies

As already noted, although ignoring namespace information allows us to deal with multiple versions of the same vocabulary, it does not allow us to deal with multiple vocabularies or with properties that change name in later versions of vocabularies. Dealing with multiple vocabularies and vocabulary versions is a common problem for the Semantic Web, so the Semantic Web architecture should provide appropriate tools to define equivalences between vocabularies on a property-by-property basis. At the time of writing, the current version of RDF Schema does not contain any method of defining equivalence, so it is necessary to use higher-level ontology languages such as DAML+OIL[15] or the forthcoming Web Ontology Language (OWL)[16]. These ontology languages can be used to describe the classes and properties used in RDF models. For example OWL provides the samePropertyAs construct that could be used to map between properties in different UAProf vocabularies in the following way:

```
<owl:ObjectProperty rdf:ID="ColorCapable">
  <owl:domain rdf:resource="#HardwarePlatform"/>
  <owl:samePropertyAs
    rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
    20000405/ColorCapable"/>
  <owl:samePropertyAs
    rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
    20010330/ColorCapable"/>
  <owl:samePropertyAs
    rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
    20020710/ColorCapable"/>
</owl:ObjectProperty>
```

In the example, a property ColorCapable is defined which is equivalent to the ColorCapable properties defined in the three versions of the UAProf vocabulary. Currently DELI does not support using OWL in this way, as the OWL specification is still being defined and there are no processors that implement OWL. It is possible to imagine that in the future OWL could help CC/PP hide some of the complexities of multiple vocabularies and multiple vocabulary versions from the application programmer.

There are some limitations on OWL's power to convert between different vocabularies. OWL can cope with some differences in encoding and describe equivalence between vocabularies that use different literals to describe the same property value. For example, ignoring device capabilities for a moment, it is possible in OWL to describe equivalence between a vocabulary which defined the property *business type* with property value *car hire* onto a vocabulary with the property *business category* with property value *automobile rental* i.e. convert between British and American business terms. However there are some differences in encoding that cannot be expressed. For example consider the situation where one vocabulary ScreenSize is the entire size of the screen in pixels (AbsoluteScreenSize) whereas in another it is the renderable area of the screen (RenderableScreenSize). In order to map between these two encodings it is necessary to have another piece of information e.g. the size of the unrenderable area and to define the mapping between the properties using a mathematical relationship that describes how to convert AbsoluteScreenSize to RenderableScreenSize. Currently languages like OWL do not allow property equivalences to be defined in this way, although they can use class-hierarchy relations to describe that there is some relationship between these properties but the properties are not identical. For example OWL could be used to create an abstract property called ScreenSize and derive two sub-properties, AbsoluteScreenSize and RenderableScreenSize. This does not necessarily help us with the task of converting one vocabulary format into another.

2.5.4 Other approaches to versioning

As already noted, previously UAProf have taken a specific approach to versioning vocabularies where the entire vocabulary is duplicated. This creates additional complexity because unchanged properties are duplicated. An alternative approach is to only include new and changed properties in new vocabulary versions. Here profiles using the updated vocabulary use the updated vocabulary and the previous version of the vocabulary concurrently, reducing unnecessary duplication. As versioning is a generic problem for the Semantic Web, perhaps there needs to be a more thorough consideration of the optimal way to version resources?

2.5.5 Core device properties

A proliferation of different vocabularies that use different, but related, properties to describe device capabilities is clearly undesirable as it creates complexity for applications developers and content authors trying to develop device independent applications. One way of avoiding this proliferation is to create a set of core device properties that are then reused in target vocabularies. This provides standardization along with flexibility as vocabularies are free to add additional properties where necessary to cope with specific devices or use cases.

This approach is similar to the approach taken in the Dublin Core initiative that defines a set of core properties for document metadata. Although there is clear value in a core set of properties, creating such a set of properties is difficult. For example how do we decide what goes into it, and what is left out? Or more importantly who decides this? Currently the W3C Device Independent Working Group work is considering this problem[17]. In the authors' opinion, this work is essential to the problem of device independence although the process of ensuring a wide adoption of these core device properties may be difficult.

3 Conclusions

This paper has described a number of approaches that can be used in CC/PP processors to simplify the use of multiple vocabularies and vocabulary versions. In the future, the Web Ontology Language (OWL) and the W3C Device Independence Working Group work on core device attributes should simplify this problem. It also makes two proposals: firstly it would be helpful if there were some general principles about how to version vocabularies to simplify processing with Semantic Web tools. Secondly, in the authors' opinion, a fundamental change in Web architecture to reflect that fact that the Web is a collection of evolving, versioned resources could help simplify the problem of dealing with multiple versions of vocabularies.

---

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

[2] *W3C RDFCore Working Group*
http://www.w3.org/RDF/

[3] Namespaces in XML
Bray, T., Hollander, D., Layman, A.
W3C Recommendation 14 January 1999
http://www.w3.org/TR/1999/REC-xml-names-19990114/

[4] Uniform Resource Identifiers (URI): Generic Syntax
Berners-Lee, T., Fielding, R., Irvine, U.C., Masinter, L.
IETF RFC2396 August 1998
http://www.ietf.org/rfc/rfc2396.txt

[5] RDF Vocabulary Description Language 1.0: RDF Schema
Brickley D., Guha, R. V.
W3C Working Draft 30 April 2002
http://www.w3.org/TR/rdf-schema/

[6] Open Mobile Alliance
http://www.openmobilealliance.org/

[7] DELI : An open source CC/PP and UAProf processor
Butler, M. H.,
HP Labs Technical Report HPL-2001-260
http://www-uk.hpl.hp.com/people/marbut/DeliUserGuideWEB.htm
See also http://delicon.sourceforge.net

[8] Architecture of the World Wide Web,
Jacobs, I.
W3C Working Draft 15 November 2002,
http://www.w3.org/TR/webarch/

[9] Some Questions and answers on CC/P and UAProf
Butler, M. H.,
HP Labs Technical Report HPL-2002-73
http://www-uk.hpl.hp.com/people/marbut/someQuestionsOnCCPP.htm

[10] Comparing URIs, URLs and URNs
Overview of materials on Web addressing, W3C Architecture Domain
July 2002
http://www.w3.org/Addressing/#terms

[11] Tag URI,

http://www.taguri.org/

[12] TAG issue metadataInURI-31 : Should metadata (e.g., versioning information) be encoded in URIs?
W3C Technical Architecture Group
http://www.w3.org/2001/tag/ilist#metadataInURI-31

[13] Cool URIs Don't change
Berners-Lee, T.
W3C Style
http://www.w3.org/Provider/Style/URI

[14] Using capability classes to classify and match CC/PP and UAProf profiles
Butler, M. H.
HP Labs Technical Report HPL-2002-89,
http://www-uk.hpl.hp.com/people/marbut/capClass.htm

[15] DAML+OIL,
http://www.daml.org/

[16] W3C Web Ontology Language (OWL) Working Group,
http://www.w3.org/2001/sw/WebOnt/

[17] DI Deliverable: definitions of properties for DI
W3C Device Independence Working Group Charter 2002
http://www.w3.org/2002/06/w3c-di-wg-charter-20020612.html#delcon-coredev