Basic Ideas for User Constraints Specification in CC/PP

Tayeb Lemlouma[1] and Nabil Layaïda[1]
[1]OPERA Project
Zirst 655 Avenue de l'Europe - 38330
Montbonnot, Saint Martin, France
Tel: +33 4 7661 5281
Tayeb.Lemlouma@inrialpes.fr Nabil.Layaïda@inrialpes.fr

Abstract

In this short document, we try to give some basic ideas about writing logical constraints for client capabilities and preferences descriptions and the way to expressing this for the CC/PP framework.

1 Introduction

By a user constraint, we mean simply a logical expression (simple or complex) that contains particular information about the user context i.e. the user preferences and the capabilities of the used device. A user constraint can express that the device is not capable to play sound, it supports playing videos, it has a limited memory or displaying screen, etc. Unfortunately, the current CC/PP doesn't provide a framework that covers a complete expressing model; this is why advanced mechanisms should be ensured to reach this objective. The expressing framework should define:

a) The pure logical side of expressing user constrains,
b) The way (structures, properties, relations between structures and properties, etc.) to describe constraints in the future CC/PP, and
c) The way to process expressions in order to facilitate the server (or the content provider) task, when receiving CC/PP client profiles.

2 Constraint Variables

The constraint variable represents a principle entity of the constraint expression. For example in a constraint like: 'the device can not play sound', the constraint variable can be 'sound capability' or 'sound capable'. So, the precedent constraint can be written like this: 'sound capability = false' or 'sound capable = false'.

A constraint variable has a type and belongs to a simple or complex structure. In a CC/PP description a simple variable should be written using a property name. Example: 'sound capability' can be represented using the property <soundCapability> or <soundCapable>. From the syntax side, the name of the property and its value type will depend to the schema in which the property belongs. The semantic of the property will depend to its path in the global profile (sometimes called *context path*). For example a 'height' variable can concern the height of the accepted images (figure 1) or it can concern the height of the device screen (figure 2).

```
<imageAccept>
<height>100</height>
...
</imageAccept>
```

Figure 1. A first use of the 'height' variable

```
<deviceScreen>
<height>320</height>
...
</deviceScreen>
```

Figure 2. A second use of the 'height' variable

Two kinds of variables are distinguished:

1- Atomic variable: A variable that can have one value of a simple type (String, Integer, Boolean, well predefined set, etc.), For example: a screen height (Integer), sound capability (Boolean), language font (a well predefined set), etc.

2- Complex variable: Can be an ordered or non-ordered set of atomic values. The set structure is used to express that the variable hasn't a unique value. If the values order is important an ordered set is used otherwise we use a non-ordered set.
Example: accepted image format = {BMP, GIF}[non-ordered set], accepted language= {French, English, Spanish}[an ordered set].

Complex variables can be presented in the CC/PP description using some existing structures such as RDF Bag for non-ordered sets and RDF Seq for ordered sets.

3 Constraints Expressions

Constraints expressions are logical expressions that use variables and values to describe a sub set of the client context. A constraint evaluation can have two possible values: TRUE or FALSE. Parsing and evaluating the user constrains is done at the server (or the proxy) side in order to deliver an adapted content to the client, this is why the user description should follow a simple and clear way.

3.1 Atomic Expressions

An atomic constrain is given as follows:
V = y, where V is the variable name and y is the value of V.
The simplest XML presentation of this constraint is:
<A>y</A>, where A is the property name of V.
Example: 'Screen width = 240 pixels' represents an atomic constraint, the XML presentation of this constraint can be: <width>240</width>

An atomic constraint can use a complex variable. Example: 'accepted image format = {BMP, GIF, SVG}'. In a CC/PP description, the XML representation of an atomic constraint that uses a complex variable requires more complex structures. RDF bag and seq represent good structures to describe constraints that use a variable in the form of ordered or non-ordered set respectively.

A simplified CC/PP description of the precedent constraint can be given as follows:

...
<acceptedImageFormats>
<Bag>
BMP
GIF
SVG
</Bag>
</acceptedImageFormats>
...

3.2 Construction of Complex Constraint Expressions

As we can see, using the above atomic constraint definition allows only to express constraints in the form of 'variable = value'. In order to describe the user capabilities more efficiently we need more advanced expressions that can describe the other logical relationships which are: '< and NOT' (less than and not), or '> and NOT' (greater than or not).

Enabling the CC/PP description of logical relationships in complex constraints can be done using different ways. We identify two description ways that can be combined:

1- *Using properties name*: Here the approach is to use a property name that contains some logical meaning of the constraint. To simplify, the following constraint: 'the wbmp card size <= 2000 Bytes' can be represented in a CC/PP profile like: <maxCardSize>2000</maxCardSize>

2- *Defining expression formats for properties value*: Here the format of the property value is defined to include logical constraints. For example: 'LE' can express the 'less or equal' relationship, and thus a constraint in the form of:

'the accepted image height <= 3000 Bytes' can be described as:
<size>
<constraint>height LE 3000</constraint>
</size>

3.3 Combining Constraint Expressions

Once a complete definition of atomic variables, expressions and the way to describe them in CC/PP using: properties, properties value and structures; mechanisms of constructing complex expressions should be defined. These mechanisms should ensure the way to express the different situations of constraint combinations. We identify the following cases:

Conjunction: E1 AND E2
Disjunction: E1 OR E2
Negation: NOT E
Exclusion: exclude (E1) from E (which is equivalent to the expression 'E AND NOT E1')
Such as E, E1 and E2 are constraint expressions.

Conjunctions can be used to describe a set of the user capabilities and the user preferences, for example concerning the device displaying capabilities of images: the accepted image format 'and' the image maximum size 'and' ... etc. The conjunction can be chosen as the default logical relationship between different constraints inside a profile (which is the case of the actual profiles), and so no new structure definition is needed. To avoid profiles ambiguity and to make the general framework more efficient, a simple structure should be defined to express conjunctions. A simple structure can be for example a AND bag element that includes different constraints. So the CC/PP description can be:

<andBagConstraints>
XML representation of constraints 1
XML representation of constraints 2
...
XML representation of constraints *n*
</andBagConstraints>

Disjunctions can be used to express that at least one of the given constraints must be satisfied. This can be utile when describing different entities inside a description element, for example to say: '(image maximal size = X *and* the format = Y) *or* (image maximal size = Z *and* the format = T)'. Similarly to the and description structure, a CC/PP presentation of the disjunctions can be using a OR bag. So the description can be in the following form:

<orBagConstraints>
XML representation of constraints 1
XML representation of constraints 2
...
XML representation of constraints *n*
</orBagConstraints>

The negation can be utile in the user description, when we have to describe that constrains are not true. This is used for example to avoid the expressing of a long list of true constraints, e.g. to say: 'the JPEG format of images is not accepted' which is equivalent to 'not (accepted image format = JPEG)'. The CC/PP description of a negation can be done using a NOT bag. The bag can contain one constraint or more, and this means that all the included constraints are related by a logical AND, and each one of these constraints is preceded by a NOT. The following describes the constraints 'not C1'.

<notBagConstraints>
XML representation of C1
</notBagConstraints>

To express 'not C1 and not C2', we can write:

<notBagConstraints>
XML representation of C1
XML representation of C2
</notBagConstraints>

Without the need of adding an andBagConstraints that includes C1 and C2. If we add the and bag the description still the same.

Using exclusions is utile to describe some constraints that aim to exclude some elements from a predefined set or schema of elements. This is useful to write short profiles by avoiding the overriding of long predefined sets. For example the exclusion can be used to express: 'the accepted modules are those of SMIL 2.0 without the animation and the content control module'. A simple CC/PP description of this can be done using an exclude bag, as follows:

```
<acceptedModules>
<modulesLanguage>SMIL 2.0</modulesLanguage>
<excludeBag>
<li>animation</li>
<li>content control</li>
</excludeBag>
</acceptedModules>
```

Elements of the exclude bag should have the same type as the global set in which we exclude elements. For example, in the above description we exclude elements from the modules set of SMIL 2.0 language, so the type here is 'SMIL 2.0 modules'.

It's important to note that we have identified the different forms of logical structures inside CC/PP profiles in order to make the profiles description easy to do by authors and efficient to handle by servers. Another solution is to define only structures for 'AND and NOT' or 'OR and NOT' relationships, since all the possible constraint expressions can be guaranteed using combinations inside these two conjunctions.

4 Conclusions

In this document we have presented some basic ideas for enabling advanced constraints expressing inside CC/PP. We have identified some basic entities to be ensured if we aim to design an efficient framework for user constraints description. Some propositions of the CC/PP description about the constraint logical form were given. However, this doesn't mean that these descriptions are the best one. Indeed, some other models can be used to write the logical form of constraints in an efficient CC/PP description (such as using DAML for disjunctions, etc.)