Icalendar Update Functions

The SOAP world requires some form of update mechanism as complete replacement is not an option. In addition an update rather than complete replacement is useful in general - for example, it avoids sending an entire event with attachments just to change the partstat.

There have been a number of proposals for relatively simple approaches and one using Xpath which is not so simple. The simple approaches generally have the problem that they still require complete replacement of parts of the event (properties, alarms), possibly losing x-parameters.

Xpath

The Xpath approach, while more complex, does allow us to target a change right down to the value of a parameter or property and does allow selection at each step in the path. Generation of Xpath is moderately complex but if built in from the start probably less so. It is also possible to diff 2 calendar entities and generate an xpath from that diff.

An Xpath based update would have the following parts:

- An href for the entity to be updated.
- A set of namespaces
- A set of updates

The namespaces are needed because of the way Xpath evaluation takes place. The updates contain xpath expressions which use namespace prefixes which must be resolved. Those namespace prefixes are not defined in the enclosing XML nor are they known at the receiving end.

Each update consists of the following items:

- Delete, add or replace indicator (may be the enclosing element)
- An xpath selector
- For add or replace a new value

The new value may be an icalendar fragment, e.g. a new property, or a new value. How this new value is used depends upon the xpath. If it selects the value element of a property for example, then the value of the new value element is used. If it selects the property itself, then the new value is used as it stands.

For example, if we select .../summary/text() we are replacing the value of the summary property. If we select .../summary we are replacing the entire summary property.

The problem with Xpath

The xpath approach works fine - up to a point. However, its implementation is very messy - even if simple. This is speaking from the Java point of view but other frameworks may be similar. The Xpath libraries in Java do a good job of allowing the evaluation and use of such an xpath update as long as the entity to be updated is in the form of an org.w3cc.dom.Document. There are at least 2 problems with this:

- Internally the event is probably not in that form so needs a conversion in both directions.
- The entire document needs validating afterwards as it's difficult to track which properties get

changed.

There are no pluggable points in the java xml apis that allow us to determine what properties or components are being changed or replaced. We can carry out some analysis of the document as we proceed and can probably derive that information but it may be a little difficult. Even though we can derive the path down to the selected element it's hard to determine what selected a given element.

A different approach.

The xpath approach may in fact be too generalized: it provides an update mechanism for an y XML document. A more calendar-centric approach would be fine and may help by providing more information to the receiver. We probably don't need the full power of Xpath predicates though we need enough to uniquely identify the component, property or parameter to be updated. Additionally, it would be useful to have a selection approach that is fully defined in the schema so that it arrives fully parsed.

It only remains then to step through the selector elements, choosing the targeted entity.

The approach I suggest is a selector which is made of a set of nested selection elements. Each element contains the appropriate entity type for the stage we are at in processing. Assuming the default namespace for the service and "x" for icalendar in xml a selection might take the following form:

```
1      <select>
         <x:vcalendar/>
2        <select>
           <x:components/>
3          <select>
             <x:vevent>
               <x:properties>
                 <x:uid><x:text>01234567@example.com</x:text></x:uid>
               </x:properties>
             </x:vevent>
4            <select>
               <x:properties/>
5              <select>
                 <x:summary/>
                 <change>
                   <x:summary>
                     <x:text>New summary</x:text>
                   <x:summary/>
                 </change>
               </select>
6              <select>
                 <x:description/>
                 <change>
                   <x:description>
                     <x:text>New description</x:text>
                   <x:description/>
                 </change>
               </select>
             </select>
           </select>
         </select>
       </select>
```

The steps then go as follows:

1. Select the nested (first?) vcalendar

2. Select the nested components
3. Select the event with a properties element containing a uid with the given value
4. Select the events properties element
5. Select the summary then change its value
6. Select the description and change its value

While this is reasonably verbose it is simple to build and fully expressed in the schema. Additional attributes to the select element would allow for modifers to the value selection, e.g. caseless, initial substring etc.

More complex scenarios

The possible VPOLL entity being discussed defines a rather more complex object we may want to manipulate. Essentially it will consist of a VPOLL entity with its own uid containing one or more other entities, tasks, events, availability etc. At the outer level we still only have one entity to select. At the next component level down we have a number of entities. Each entity can be identified by its uid and possibly a recurrence id.

We will probably run into similarly complex objects with the interval and gluon approach for the smart grid work.

Selecting master, instance or all.

We need to be able to manage the following selections:

1. Master only
2. Instance only
3. Master and all instances

If we assume we only need to support selection by uid and/or recurrence id we can manage case 2 by providing the uid and recurrence id. If we provide the uid only do we intend case 1 or 3? I suggest the following approach:

1. Master only - uid is provided
2. Instance only - uid and recurrence id is provided
3. Master and all instances - uid only with all=true attribute on select element.

An alternative approach is to disallow the selection of master and all instances. Updates to overrides require a series of updates. There may be a lot of them but they can (and should) all be carried in a single request.

Note we can allow the simple case of no selection properties for a non-recurring entity.

Further Issues

In the select element above we have shown single valued properties only. However, we will need to be able to select a given property - perhaps for deletion. Some can be selected on the basis of their current value, e.g. category. However, other properties and components may be more difficult. To simplify the implementation it may be worthwhile specifying what properties or values are used to match on when applying updates.