Tablix2 file format – looking forward

Tomaž Šolc
30.1.2008

This discussion document contains a short description of the XML file format used by Tablix from 0.3.0 version on (the also called Tablix2) with suggestions for future file formats based on similar timetabling models.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ttm PUBLIC "-//Tablix//DTD TTM 0.2.0//EN"
"http://www.tablix.org/releases/dtd/tablix2r1.dtd">
<ttm version="0.2.0">
```

Root XML node, containing file format version and document type definition.

```
<info>
      <title>Title</title>
      <address>Address</address>
      <author>Author</author>
</info>
```

File format defines these tags with meta data. They are not touched by the algorithm and are only used when exporting a timetable to other formats.

Future formats could include support for fields (for example processing date and time, information about program name and version, etc.)

```
      <modules>
            <module name="sametime.so" weight="60" mandatory="yes"/>
            <module name="timeplace.so" weight="60" mandatory="yes"/>
            <module name="holes.so" weight="60" mandatory="yes">
                  <option name="resourcetype">teacher</option>
            </moule>
      </modules>
```

This part defines timetabling constraints. It tells Tablix what is considered a feasible timetable and what is considered a good timetable by specifying a fitness function.

Tablix uses numeric functions compiled to binary modules as parts of the fitness function – this is clearly unsuitable to be used in a standard that is meant to be supported by more than one application. Some kind of a scripting language may be used instead.

```
      <resources>
            <constant>
                  <resourcetype type="teacher">
                        <resource name="a">
                              <restriction type="conflicts-with">b</restriction>
                        </resource>
```

```
                        <resource name="b"/>
                        <resource name="c"/>
                        <resource name="d"/>
                  </resourcetype>
                  <resourcetype type="class">
                        <resource name="1"/>
                        <resource name="2"/>
                        <resource name="3"/>
                  </resourcetype>
            </constant>
            <variable>
                  <resourcetype type="room">
                        <linear name="#" from="1" to="100"/>
                  </resourcetype>
                  <resourcetype type="time">
                        <matrix width="5" height="7"/>
                  </resourcetype>
            </variable>
      </resources>
```

This part defines two logically separate things:

It tells the type of the problem being solved by specifying resource types. Resources are generalized objects that are assigned to events. Tablix itself doesn't care what each resource type represents – it is the fitness function definition (i.e. binary modules) that makes that distinction.

Kernel only cares about one property – if a resource type is variable or constant. Constant types are statically assigned to events by the user (and so form the a part of the problem definition), while assignment of variable types depends on Tablix (and forms the solution to the problem). Support for more generalized resource type properties may prove useful – like specifying that a certain resource type represents people or groups of people. This would allow reuse of parts of fitness functions specialized in certain type of resources.

In addition to the type this part also defines the actual resources. That is for example names of teachers and classes (groups of students) and also rooms and time slots. Example above demonstrates two shortcuts in defining resources: <linear> tag defines an array of 100 resources with names from „1“ to „100“. <matrix> tag defines a matrix of 35 resources with names from „0 0“ to „4 6“

These two logical parts would better be separated. There are some cases even when it would be nice to have them in separate files (like an graphical user interface where you could load a schema and the interface would change according to what you loaded)

<restriction> tags are a way to assign generic properties to resources which can affect the fitness function. These properties can include such things as room capacity, capability of rooms to hold certain types of events, restrictions regarding teacher availability, number of students in a group, etc.

```
    <events>
        <event name="1" repeats="1">
            <resource type="teacher" name="a"/>
            <resource type="class" name="1"/>
        </event>
        <event name="2" repeats="1">
            <resource type="teacher" name="b"/>
            <resource type="class" name="2"/>
        </event>
        <event name="3" repeats="1">
            <resource type="teacher" name="b"/>
            <resource type="class" name="3"/>
        </event>
        <event name="4" repeats="1">
            <resource type="teacher" name="c"/>
            <resource type="class" name="1"/>
        </event>
        <event name="5" repeats="1">
            <resource type="teacher" name="d"/>
            <resource type="class" name="3"/>
        </event>
    </events>
```

This part defines assignments of resources to events. Note that the format used for problem definition is also used to store the solution. The only difference is that problem definition only includes assignments of resources of constant resource types while problem solution includes assignments of all resources. User can also assign variable resources in the problem definition to give the algorithm a starting point (e.g. you can use the output of one Tablix run as an input for another).

Tablix defines events as tuples of exactly one resource of each type. This is the format that is best suitable for solving with a genetic algorithm.

Variable number of attendants to one event can be simulated in a couple of ways (for example having multiple logical events as one physical in the fitness function). Similar solutions exist for events that span more than one time slot.

This basic decision made format less understandable and complicated fitness function definitions. On the other hand it had the benefit that it made the core timetabling model is simpler and also greatly simplified kernel code.

Events, like resources, can also have their own <restriction> tags which add generic properties to events.

Repeats property is a shortcut. For example an <event> with repeats set to 5 is equivalent to five separate, identical <event> tags (in contrast to resources, name property of events doesn't need to be unique).

```
</ttm>
```

In conclusion there are five logically distinct blocks of data stored in a Tablix XML file:

- Problem type definition (resourcetypes)
- Problem definition
    - constraints definition (modules, restrictions)
    - variable domain definitions (resources)
    - known values (constant resources in events)
- Problem solution (variable resources in events)

The format sacrifices the clear distinction between these parts for a more compact file, which is easier to write and read by hand. Particularly the use of same file to specify problem definition and solution turned out to be practical.

Tablix also sacrifices some of the file format's flexibility because of implementation details and performance:

- Fitness functions defined as binary modules
- Number of resources in an event is fixed

Last thing to note would be that the format is inheritly oriented around discrete problems. That is events can not occupy a fractional number of time slots. This makes it unsuitable for solving certain classes of timetabling problems where some variable (usually time) can take a large number of values.

See also:

- Tablix timetabling model, format description
  http://tablix.org/~avian/ttm2.pdf
- Tablix User's Manual, Configuration file format
  http://www.tablix.org/releases/doc/manual.html/x360.html